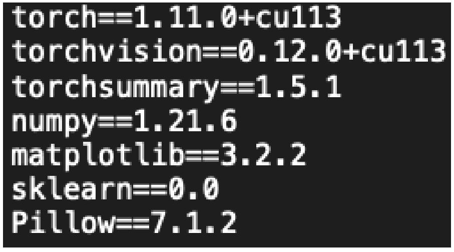
☆ Framework: pytorch

☆ Requirements: see **requirements.txt**



```
torch==1.11.0+cu113
torchvision==0.12.0+cu113
torchsummary==1.5.1
numpy==1.21.6
matplotlib==3.2.2
sklearn==0.0
Pillow==7.1.2
```

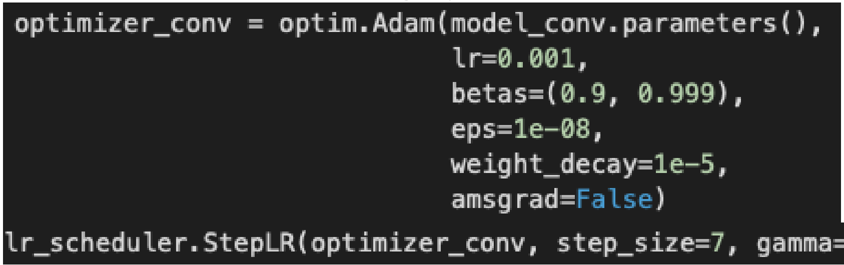
☆ Model architecture: Resnet152

The original Resnet architecture is shown in **resnet_arch.png**.
Resnet152 is its variant with a larger size and more layers.
The fully connected layer is changed to output size of 10. The final architecture can be viewed in **torch_summary.txt.**

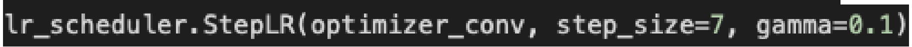
☆ Model source: **Pretrained model resnet152** from pytorch

☆ Hyperparameters:

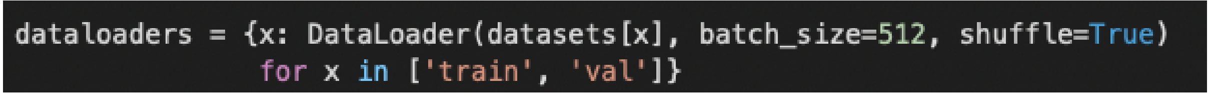
- Adam optimizer with learning rate (0.001) and weight decay (1e-5)
- Cross Entropy Loss
- learning rate scheduler (Decays the learning rate of each parameter group by gamma every step_size epochs.)
- batch size 512
- trained without freezing layers.



```
optimizer_conv = optim.Adam(model_conv.parameters(),
                            lr=0.001,
                            betas=(0.9, 0.999),
                            eps=1e-08,
                            weight_decay=1e-5,
                            amsgrad=False)
```



```
lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

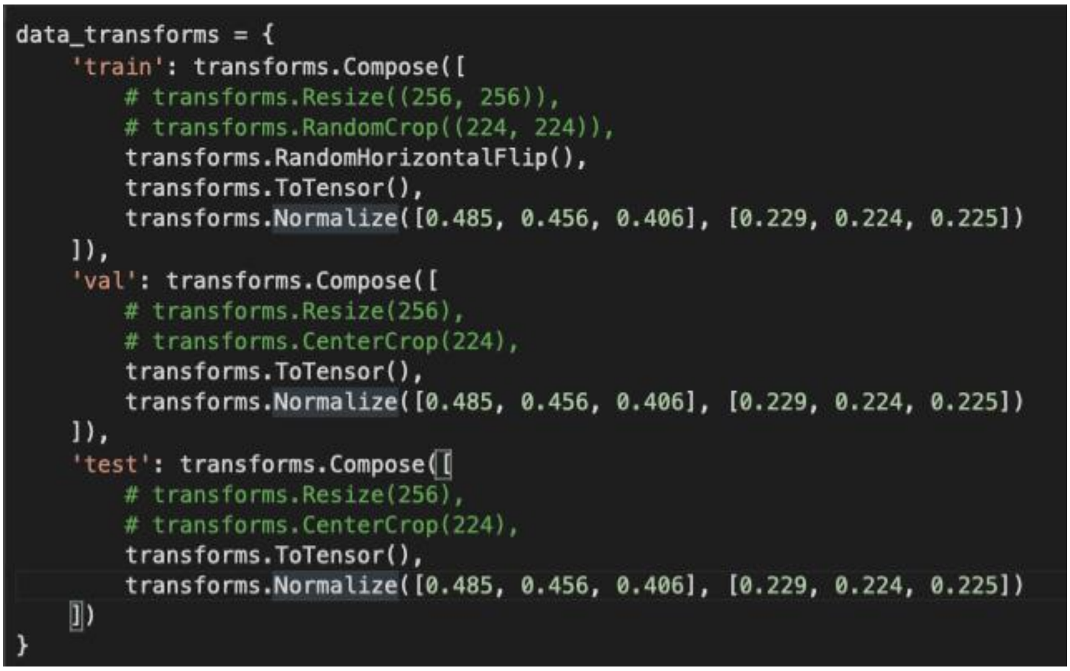


```
dataloaders = {x: DataLoader(datasets[x], batch_size=512, shuffle=True)
               for x in ['train', 'val']}
```

☆ Data size: Training data size 47000, validation data size 3000

☆ Data preprocessing & augmentation:

- train: random horizontal flip, normalization
- val: normalization
- test: normalization



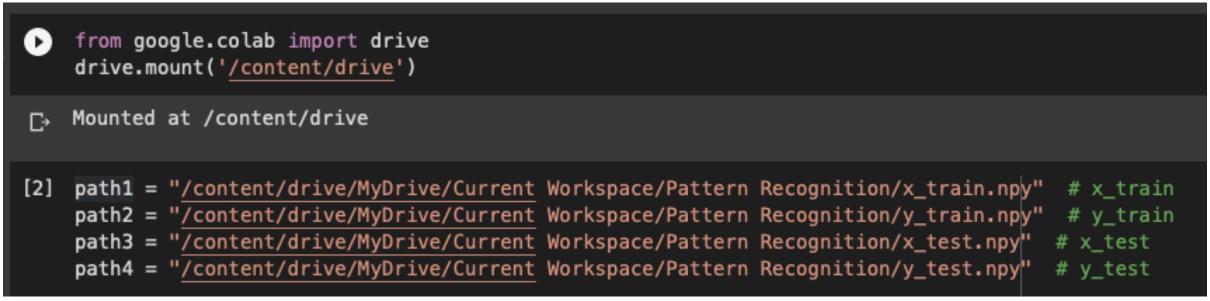
```python
data_transforms = {
    'train': transforms.Compose([
        # transforms.Resize((256, 256)),
        # transforms.RandomCrop((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        # transforms.Resize(256),
        # transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        # transforms.Resize(256),
        # transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
}
```

☆ Environment & organization:

All source code is contained in **0811521_HW5.ipynb** and **inference.ipynb**, which I run at Colab. The model weights and data are loaded from google drive (need authentication).

0811521_HW5.ipynb:

This will import pretrained model, and then train and evaluate the model. The paths for data can be configured at the beginning.



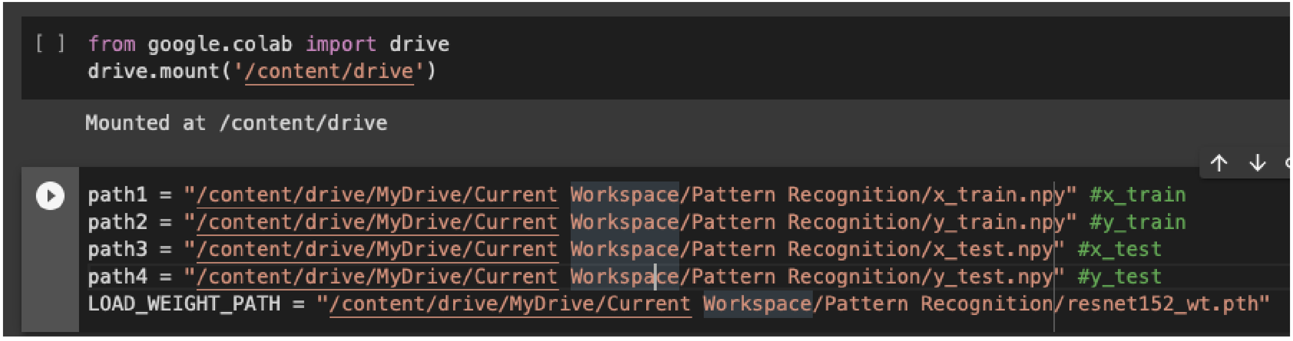
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
path1 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/x_train.npy"  # x_train
path2 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/y_train.npy"  # y_train
path3 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/x_test.npy"  # x_test
path4 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/y_test.npy"  # y_test
```

inference.ipynb:

I have trained the model and stored its model weights. **inference.py** will load the weights and evaluate the model. The model weights are stored in **resnet152_wt.pth.** The paths for data and model weights can be configured at the beginning.

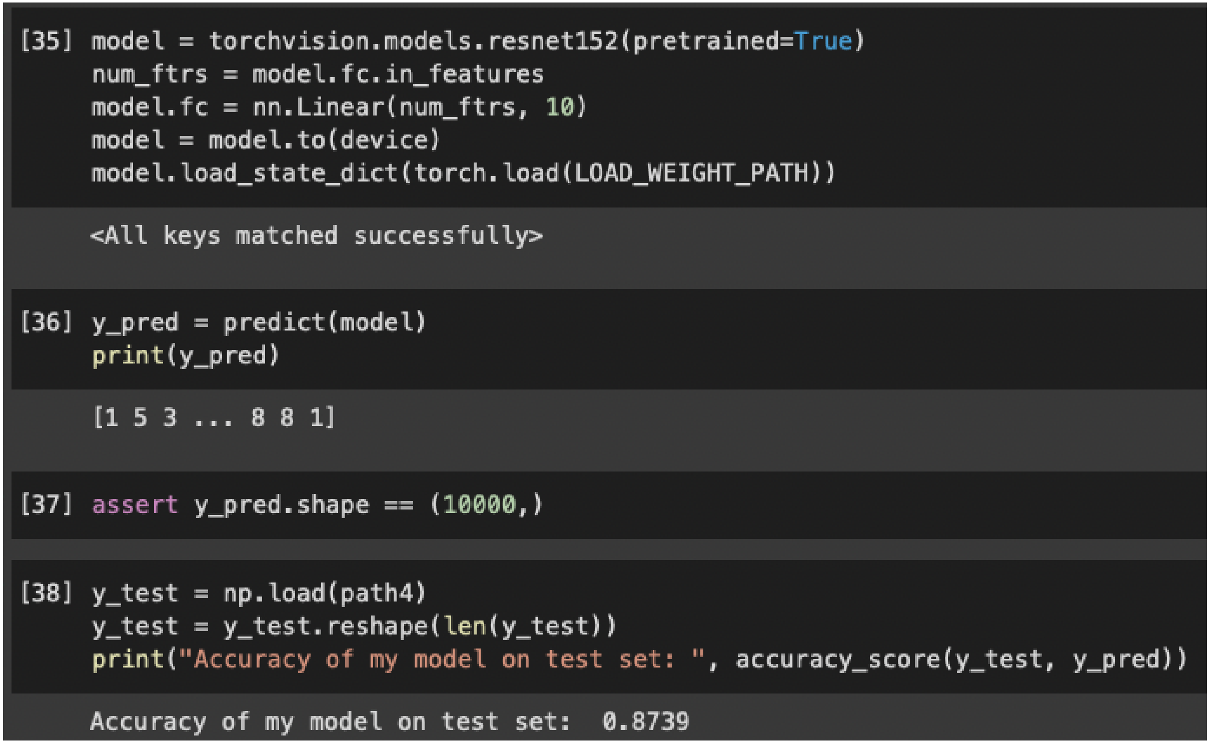


```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
path1 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/x_train.npy" #x_train
path2 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/y_train.npy" #y_train
path3 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/x_test.npy" #x_test
path4 = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/y_test.npy" #y_test
LOAD_WEIGHT_PATH = "/content/drive/MyDrive/Current Workspace/Pattern Recognition/resnet152_wt.pth"
```

☆ Result & Accuracy:



```
[35] model = torchvision.models.resnet152(pretrained=True)
     num_ftrs = model.fc.in_features
     model.fc = nn.Linear(num_ftrs, 10)
     model = model.to(device)
     model.load_state_dict(torch.load(LOAD_WEIGHT_PATH))

     <All keys matched successfully>

[36] y_pred = predict(model)
     print(y_pred)

     [1 5 3 ... 8 8 1]

[37] assert y_pred.shape == (10000,)

[38] y_test = np.load(path4)
     y_test = y_test.reshape(len(y_test))
     print("Accuracy of my model on test set: ", accuracy_score(y_test, y_pred))

     Accuracy of my model on test set:  0.8739
```